Collision Detection for Rigid Bodies: A State of the Art Review

Charbel Fares[1,2] and Yskandar Hamam[1,2]
[1]LIRIS, CNRS FRE 2508, 10-12 avenue de l'Europe, 78140 Velizy, FRANCE
[2]ESIEE, Lab. A[2]SI, Cité Descartes, BP 99, 93162 Noisy-Le-Grand, FRANCE
{c.fares, y.hamam}@esiee.fr

Abstract

Virtual reality applications refer to the use of computers to simulate a physical environment in such a way that humans can readily visualize, explore, and interact with "objects" in this environment. The design of virtual scenes requires realistic physically based simulation algorithms and in particular efficient collision detection routines. Collision detection prevent penetrations between objects.

This paper presents an overview of the collision detection procedures for rigid bodies. It divides those procedures in two families that form a general hybrid collision detection algorithm. The first family is used in the first phase known as broad phase that roughly detects collision. The second family is used in the second one known as narrow phase that exactly detects if collision occurs.

Keywords: *Collision Detection, Rigid Bodies, Virtual Reality, Bounding Volumes Hierarchies, Penetration Depth Vector.*

1 Introduction

Designing an efficient collision detection system is a bit like preparing a soup. The final will be a mixture of different sub-systems. The importance of those sub-systems is that when they are used alone they can produce a similar effect to the final system but with less resolution or slower rate.

A virtual environment is a computer-generated world filled with virtual objects. Such an environment should give the user a feeling of presence, which includes making the images of both the user and the surrounding objects feel solid. For example, the objects should not pass through each other, and things should move as expected when pushed, pulled or grasped. Such actions require accurate collision detection, if they are to achieve any degree of realism. However, there may be hundreds, even thousands of objects in the virtual world, so a naive algorithm could take a long time just to check for possible collisions as the user moves. This is not acceptable for virtual environments, where the issues of interactivity impose fundamental constraints on the system. A fast and interactive collision detection algorithm is a fundamental component of a complex virtual environment. Ensuring objects interact in the correct manner is very computationally intensive and much research has addressed the issues involved with trying to reduce the computational requirements by simplifying the representation of the objects in the scene.

The range of applications that require collision detection is extensive. Vehicle simulators are one case where the users manipulate a steering device, and attempt to avoid obstacles in their path [1]. In molecular modeling, simulation allows interactive testing of new drugs to examine how molecules interact and collide with each other. Training and education systems that realistically model the movement of objects within the geometric constraints of their layout, allow designers to experiment interactively with different strategies, e.g. to assemble or disassemble equipment, to perform a virtual surgery [2]-[4], or to test different paths that a robot could take [5]. Such simulations are a safe provide low cost training means. Human characters animation is one of the most challenging topics in computer animation, and in this case collision detection is an important issue. Collision must be detected between the virtual person and their environment [6], clothes [7] and hair [8]. Self collision should be tested as well.

This paper addresses the following problem: Given a virtual environment, E, in three dimensions, and a moving object A; preprocess E and A into a data structure of small size so that queries of the form, "Does any of the sub-object A intersect any of the sub-parts of E?" can be answered very rapidly. Furthermore, a study of the problem of tracking the motion of A within E is done in order to detect dy-

namically, in real time, when A collides with E.

When animating more than two objects, the most obvious problem which arises is the $O(N^2)$ problem of detecting collisions between all N objects. It is obvious that this is an undesirable property of any collision detection algorithm, and several techniques have been proposed to deal with it such as the hybrid collision detection approach. This approach refers to any collision detection method which first performs one or more iterations of approximate tests to identify interfering objects in the entire workspace and then performs a more accurate test to identify the object parts causing interference. Thus to solve this problem, a hybrid collision detection approach is used to tackle the problem in various phases. The initial phase of such an approach, known as the broad phase, aims to efficiently cull out pairs of objects which cannot possibly be interacting. A number of different techniques could be used to achieve this coarse grain detection such as Sweep and Prune [9], global bounding tables [10] or overlap tables [11]-[13]. Having determined which objects are potentially interacting, the hybrid approach a finer grained algorithm is used to narrow in on the regions of objects which are in contact. This narrow phase processing typically traverses hierarchical representations of the objects to hone in on the regions of interest. The broad phase reduces the amount of work that is required to perform the narrow phase. Algorithms such as those of Lin-Canny [14], V-Clip [15], I-Collide [16] or enhanced GJK [17] may be used in the narrow phase.

The rest of the paper is structured as follows. In section 2 factors that affect the collision detection design is shown. Section 3 discuses the different algorithms used in the broad phase, while section 4 describes algorithms used in the narrow phase. The penetration depth computation is shown in section 5. Finally, conclusions is sketched in Section 6.

2 Collision Detection Design Issues

Different factors affect the design of a collision detection system. They may be clasified into the following categories: Object representations; Type of queries; Number of objects.

2.1 Object Representations

Most current system uses triangles as the fundamental rendering primitive. Consequently a polygonal representation is a nature choice for scenes. The most generic polygonal representation is the polygon soup. It is an unordered collection of polygons with no connectivity information. Such information helps finding the "inside" of an object. They include information mentionning which edges connect to what vertices and whether the object forms a closed solid or if he is convex or concave. Adding those informations will form a larger polygonal surface called polygonal mesh. Building objects from a collection of polygon meshes is one of the most common methods for authoring geometrical models.

Representing polygonal objects may be done implicitly or explicitly. When it is defined in terms of vertices, edges, and faces, it is an explicit representation. However when it is defined throught a mathematical expression like spheres, cones, cylinders or ellipsoids, it is an implicit representation.

Collision detection between implicit objects is quick and since they are represented by known equations. Thus they may be used as rough approximations of scene objects for quick rejection culling.

2.2 Types of Queries

The most straightforward collision query is the intersection testing. It generate a boolean answer of wether two objects are colliding or not. However it is sometimes not enought to know if objects are colliding but also the intersecting parts must be found. Determining a set of contact points, known as the contact manifold, is a difficult problem. In some applications such as games, approximate queries may be sufficient. Approximate query consist of formulating the problem up to a given degree of tolerance.

If objects penetrate, one may need to find the penetration depth value. The penetration depth of a pair of intersecting objects is the shortest vector over which one object needs to be translated in order to bring the pair in touching contact.

2.3 Number of Objects

In a scene of n objects, $O(n^2)$ pairwise tests may be needed to perform the collision between all objects. Due to the quadratic time complexity, naively testing every object pair for collision quickly becomes too expensive even for moderate value of n. To reduce this number of pairwise test, a separation of the collision handling in two phases may be done. The resulting algorithm is known as a hybrid algorithm composed of a broad and a narrow phase. The broad phase identifies smaller groups of objects that may be colliding, while the narrow phase constitues the pairwise tests within subgroups. Figure 1 illustrates

how broad phase processing reduces the work load through a divide and conquer strategy.

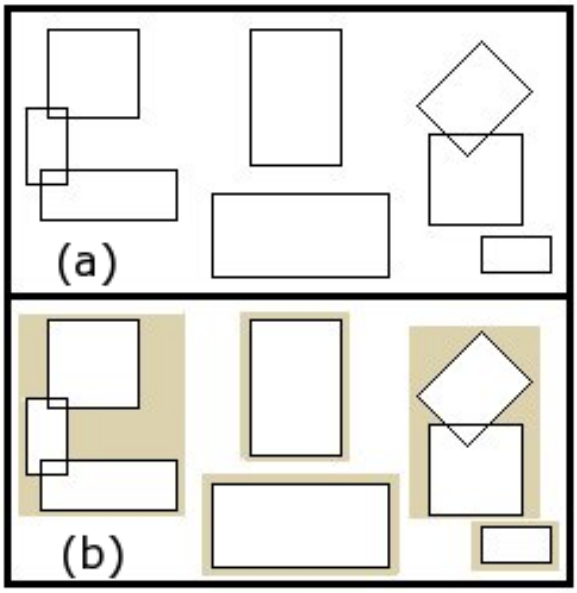

Figure 1: The broad phase identifies disjoint groups

3 Broad Phase: Refinement Level

The broad phase of a collision detection algorithm is often based on using bounding volumes and spatial decomposition techniques in a hierarchical manner. Hierarchical methods have the advantage that as a result of simple tests at a given point in the object hierarchies, branches below a particular node can be identified as irrelevant to the current search and so pruned from the search.

3.1 Octrees

Octrees are built by recursively sub-dividing the volume containing an object into eight octants, and retaining only those octants that contain some part of the original object as nodes in the tree [18]. Such a data structure is simple to produce automatically, and lends itself to efficient and elegant recursive algorithms. The disadvantage of this approach is that each level of the hierarchy does not fit the underlying object very tightly.

3.2 Sphere-Trees

Since spheres are rotationally invariant, it is very fast to update them. It is very simple as well to test for distances between them, and test for overlaps. Those were the major advantages of using such trees. The disadvantage is that spheres do not approximate certain types of objects very efficiently. In order to improve the efficiency one can build first a medial axis surface, which is a skeleton representation of an object, and then placing the spheres upon this to provide a tighter-fitting approximation to the object. This approach was used in [19]. Figure 2 shows a bounding sphere of a wrist bone.

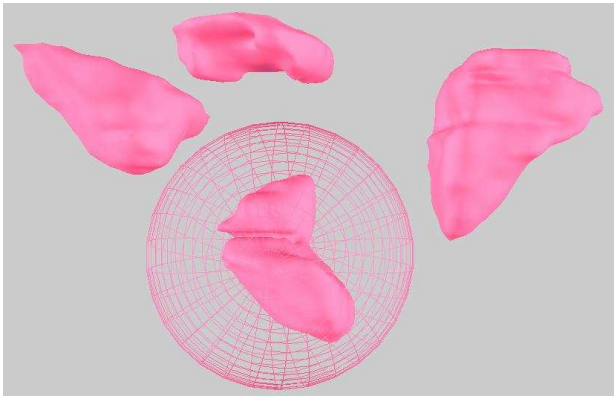

Figure 2: Sphere Bounding Boxes of the Wrist Bones

3.3 C-Trees

It consists of a mixture of convex polyhedra and spheres [20]. This has the advantage of choosing primitives that best approximate the enclosed object, but a major drawback is that the hierarchy must be created manually.

3.4 Axis Aligned Bounding Boxes (AABB)-Trees

This algorithm is the simplest one that could be used in the broad phase [21]-[22]. For constructing the AABBs, one need first to find the minimum and maximum point orthogonally projected onto the x, y and z axis. Then, with these projections, intervals are formed on each axis for each object. If different objects exist in the scene, three lists are constructed, one list for each dimension. Each list contains the value of the endpoints of the interval on the corresponding axes. By sorting each list, the corresponding pair of objects that are in contact may be determined. If two objects are in collision, their corresponding AABBs

are also in collision. That is, two AABBs are in collision if and only if their intervals along each axis overlap. If the sum of the facets of the pair of objects in collision is more than a certain threshold, the algorithm divides the AABB into four subsets of AABBs and considers four different objects. This step is done by comparing the objects offline and it could be done online as well. This process is only needed for huge rendered objects in the scene that usually their rendering time is much greater that their collision detection test time.

Two kind of AABB exists: Fixed and dynamic size. With the fixed size AABB, boxes may be generated by taking the maximum and minimum of the object with all the possible orientations. The fixed size bounding boxes are simpler because there is no need to recalculate the new maximum and minimum of objects in each step movement. However, in some cases of scene configuration, they are too much bigger than the objects, and as a consequence, they will be always in collision between each other. This case could be faced mostly with objects of longitudinal shape. Figure 3 shows a fixed size AABB. The dynamic size AABB needs to be updated dynamically at every time step.

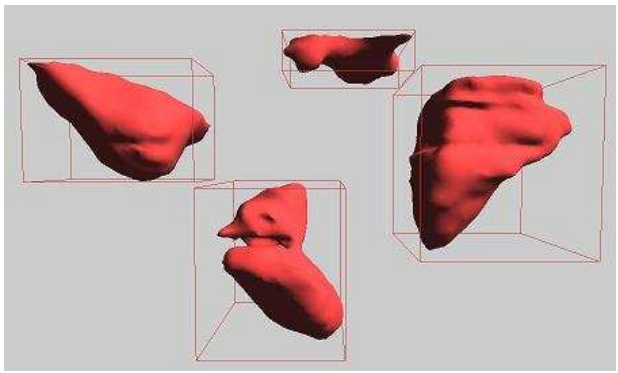

Figure 3: AABB of the Wrist Bones

3.5 Oriented Bounding Boxes (OBB)-Trees

The OBB-Tree is a hierarchical representation using Oriented Bounding Boxes (OBBs) [11]-[12]. An OBB is a rectangular bounding box at an arbitrary orientation in three dimensional space. In an ideal case, the OBB would be oriented such that it encloses an object as tightly as possible. In other words, the OBB is the smallest possible bounding box of arbitrary orientation that can enclose the geometry in question. This approach is very good at performing fast rejection tests. The disadvantages of OBB-trees over Sphere trees is that they are slower to update and orientation sensitive.

3.6 K-DOPs

This approach uses hierarchies of Discrete Orientation Polytopes, which are convex polytopes whose facets are determined by half spaces whose outward normals come from a small fixed set of k orientations [10]. Again, they implement it with a small number of highly complex objects, for the purposes of haptic force-feedback. If there is a large number of objects between which fast rejection or acceptance is needed, the update time needed for these approximations is likely to add an unacceptable additional burden. This approach is a generalization of AABBs (which are actually 6-dops), and therefore also suffers from the need for dynamic updating of the nodes.

3.7 Swept Sphere Volumes (SSV)

A swept sphere volume is a sphere that is swept out along a geometric primitive, such as a point (a sphere) line (a cylinder with rounded ends) or rectangle (a cube with rounded edges and corners) [23]. These volumes provide a means varying the shape of the bounding primitive to achieve a tighter fit to the underlying geometry, without the disadvantage of having to compute them by hand. Although they have similar performance results to OBB-trees, SSV have potentially better fit and this leads to more accurate collision tests, especially if no exact testing is performed at the end of the narrow phase, e.g. in the case of interruptible collision detection.

3.8 Minimum Volume Ellipsoidal Fitting

Many algorithms exist for constructing the minimum volume ellipsoidal fitting of a set of points [24]-[26]. An ellipsoid $E(c, A)$ having minimum volume and containing a set χ of m points could be represented by the following equation where c is the center:

$$\{x \in \Re^d \quad | \quad (x-c)^T A(x-c) \leq 1\} \qquad (1)$$

$$with \quad \chi = \{x_i, i = 1..., m \quad | \quad m \in \Re^d\}$$

Thus solving for A and c gives an ellipsoid that contains χ. The main idea is to remove form χ points by cardinality reduction. Figure 4 shows the ellipsoidal envelops of the wrist bones while figure 5 shows the second level of the ellipsoidal tree.

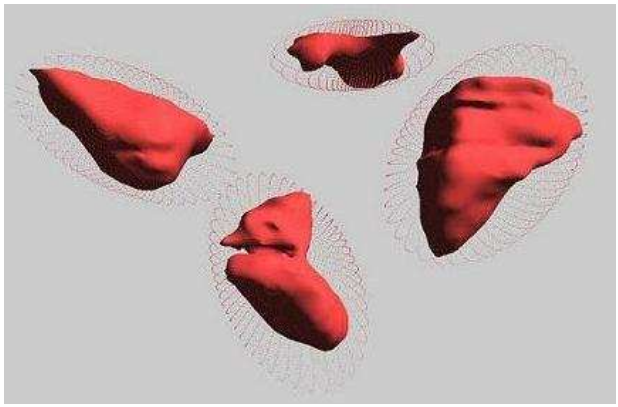

Figure 4: Ellipsoidal Envelop of the Wrist Bones

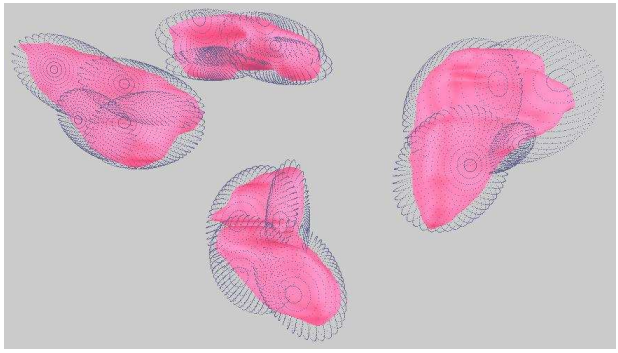

Figure 5: Second level of the Ellipsoidal Tree

3.9 Other Bounding Volumes

In addition, many other types of volumes have been suggested as bounding volumes. These includes Shell-Trees [27], cones [28]-[29], cylinders [30], and zonotopes [31]. The Shell-Tree consist of oriented bounding boxes and spherical shells, which enclose curved surfaces such as Bezier patches and NURBS. Zonotopes are centrally symmetric polytopes of certain properties. These shapes have not found widespread use as bounding volumes, in part due to having expensive intersection tests. For this reason, they are not covered in this overview.

4 Narrow Phase: Exact Level

The narrow phase of the hybrid algorithm tests whether an impact point exists between a pair of objects and specifies the point of intersection.

4.1 Linear Programming Approach

In this approach, the collision detection problem is first formulated as an optimization problem and then solved by using Linear Programming [21], [32]. It is a powerful and quick algorithm but it must be applied on a pair of convex objects at a time. Hence, in this phase, each pair of convex objects in the scene is treated separately. For non convex objects an algorithm of decomposition into convex polyhedra should be used first and then each convex part will be considered as an object.

In the formulation process, each facet i is represented by their planes inequality in the form of $a_i x + b_i y + c_i z \leq d_i$. Any point lies in the object must verify all the inequalities of the planes which formed this object. The set of formulated inequalities represents the constraints of the optimization problem. Having this formulation, if a point verifies two sets of inequalities simultaneously, then this point belongs to their corresponding objects and so a collision is detected at that point between those two objects. Physically, the "less or equal" indicates the existence of material inside the object and this mean that the facets do not only separate two region in space but they also label those separated regions. In addition to the inequality constraints, a cost function is formulated to construct the optimization problem. The cost function as well as the type of optimization (minimization or maximization) is not important for this application. For simplicity of implementation, this paper assumes maximizing an objective cost function in the form $x + y + z$. Hence, the collision detection problem may be formulated as follows:

$$\begin{array}{c} Maximize \;\; c^T \chi \\ Subject\ to \;\; A\chi \leq b \end{array} \tag{2}$$

Where χ=$[x, y, z]^T$ is the vector of variables to be solved for, A is a matrix of known coefficients and b and c are vectors of known coefficients:

$$A = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ . & . & . \\ . & . & . \end{bmatrix}, \quad b = \begin{bmatrix} d_1 \\ d_2 \\ . \\ . \end{bmatrix}, \tag{3}$$

$$c = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^T$$

With the previous scenario, the number of constraints is equal to the sum of the facets in both objects, which is usually very huge. In order to reduce the size of the problem constraints and decrease the run time, the proposed algorithm solves the dual problem. The general form of the dual problem may be represented as follow:

$$\begin{array}{c} Minimize \;\; b^T \lambda \\ Subject\ to \;\; A^T \lambda \geq c \end{array} \tag{4}$$

In the dual problem, the number of constraints becomes constant (three) while the number of dual variables λ varies depending on the original number of the planes. This procedure can be applied only on convex objects. Therefore, objects in the scene should be divided into convex sub-objects.

4.2 Medial Axis

Let $\Re^3$ be the Euclidean three-dimensional space, and d, the Euclidean distance. Let X be an object in $\Re^3$. The skeleton $S_k(X)$ of an object X, is the location of the centers of maximal spheres included in X. A sphere B included in X is said maximal, if there exists no other sphere included in X and containing B [32]-[33]. The skeleton possesses many attractive mathematical properties such as reversibility, homotopy, and invariance through translations and rotations. The reversibility property, which allows recovering the whole 3D volume from its skeleton, is a crucial assumption in this method. The most attractive property of the spheres is the rotational invariance: A sphere is invariant to the rotation of the objects and therefore its update cost is very small; no matter what type of motion the bodies are going through the spheres can be updated by simply translating their centers. The reconstructed object from the medial axis is then a set of overlapping circles. Those circles cover exactly the entire object. In order to detect collision between two circles one can simply verify the positions of their centers (c_i, c_j). A collision is occurring if the following equation is verified:

$$X_{c_j} - X_{c_i} \leq r_i + r_j \tag{5}$$

$$Where \;\; X \;\; = \;\; [x, y, z]$$

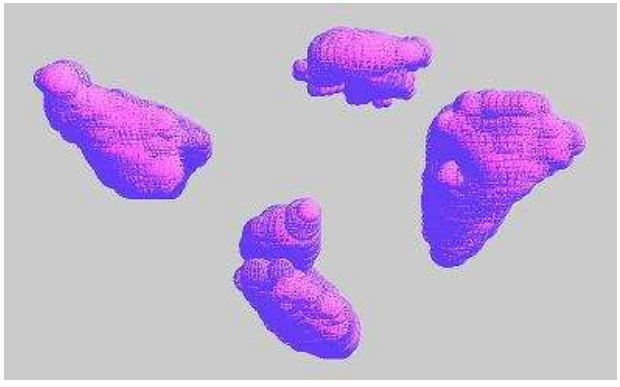

Figure 6: Medial Axis View of the Wrist Bones

4.3 Lin-Canny Algorithm

In practice, the objects that we deal with are often in continuous motion. A standard way to handle this situation is to discretize time, and at each time step, compute the distance between the objects according to their current position and orientation. If the time step is small enough, the closest pair of features (vertices, edges or faces) between two polyhedra will not move very far from one time step to the next. This coherence, together with convexity, motivates the approach of tracking the closest pair of features instead of computing it from scratch at every time step. Lin and Canny proposed the first algorithm (LinCanny) that exploits coherence [14]. The algorithm starts from the closest pair computed in the last time step, and "walks" on the surface of polyhedra until reaching the new closest pair. Convexity guarantees that one can determine locally whether a pair of features is the closest pair, and if not, a neighboring pair that is closer. Once those features are known, the distance between two polyhedra is found and a collision is declared when this distance falls below some ϵ.

The Lin-Canny algorithm does not handle penetrating polytopes, however, and if such a condition arises the algorithm enters an infinite loop. A possible solution to this problem is to force termination after a maximum iteration, and return a simple result stating that the objects have collided. However, this solution is quite slow, and no measure of inter-penetration is provided. Inter-penetrating objects will occur very frequently unless they are moving quite slowly, and/or if the detection time-step is quite small. This is unlikely to be the case in real-time applications such as games and Virtual Environments. If inter-penetration occurs, and more information is needed about the exact time of contact, backtracking is necessary to pinpoint the exact instant in time when collision occurred, a slow and cumbersome process.

4.4 V-Clip Algorithm

The V-Clip [15] tracks closest pairs of features similarly to the above-mentioned Lin-Canny algorithm. It handles penetrating polyhedra. This capability allows the algorithm to be extended to non convex polyhedra by representing them as groups of convex polyhedra. Objects may be decomposed using a convex decomposition technique [22].

4.5 I-Collide Algorithm

The I-Collide is an interactive and exact collision detection library for large environments composed of

convex polyhedra. Non-convex polyhedra may be decomposed into sets of convex polyhedra, which may then be used with this library. I-Collide exploit coherence (the property of a simulation to change very little between consecutive time steps) and the properties of convexity to achieve very fast and exact collision detection [16].

4.6 GJK algorithm

The GJK algorithm is a Simplex-based algorithm. A simplex is the generalization of a triangle to arbitrary dimensions. The approach in these cases is to treat a polytope as the convex hull of a point set. Operations are then performed on the simplex defined by subsets of these points [17]. The main strength of this algorithm is that, in addition to detecting whether two objects have collided or not, it can also return a measure of interpenetration. This algorithm achieves the same almost-constant time complexity as Lin-Canny, while eliminating most of its main weaknesses.

5 Penetration Depth Computation

The penetration depth of a pair of intersecting objects is the shortest vector over which one object needs to be translated in order to bring the pair in touching contact [34]. Methods for computing the penetration depth are less common than those of collision detection. This section shows a novel method to calculate this penetration depth vector.

Since the time step is small enough, one can approximate all the movements in the scene to be locally translational without any LOD degradation. The movement of the objects in the scene is considered to be with constant velocity. Having this in hand, one can reformulate the problem stated below in section 3.1. A variable t representing the time will be added to the system and χ will be equal to $[x, y, z, t]^T$. To solve the problem one should minimize t. Thus problem 3 becomes as follows:

$$A = \begin{bmatrix} a_1 & b_1 & c_1 & e_1 \\ a_2 & b_2 & c_2 & e_2 \\ . & . & . & . \\ . & . & . & . \end{bmatrix}, \quad b = \begin{bmatrix} d_1 \\ d_2 \\ . \\ . \end{bmatrix}, \quad (6)$$

$$c = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T$$

Then, the problem is solved like as in section 4.1 and the result will be the penetration depth vector.

6 Conclusions

Virtual reality is becoming nowadays the most important tool used for testing and training. Since it is expected to have an exact simulated environment, collision detection is an essential issue that should be treated and respected. In this paper an overview of collision detection algorithms was shown. Those algorithms were classified into two phases. The majority of the collision detection algorithm are dependent on the input data of the scene, i.e. the form that objects are represented in the environment.

Bounding volumes are simple geometric shapes used to encapsulate one or more objects of greater geometrical complexity. Most frequently, spheres and axis aligned boxes are used as bounding volumes. If a really tight fit is required, oriented boxes or convex hull may be used. Bounding volumes are used as easy overlap rejection tests, before more expensive tests are performed on the geometry enclosed within them. As discussed in this paper, there are trade-offs involved in the selection of bounding volumes shapes. By using bounding volumes of tighter fit, the chance of early rejection increases, but at the same time the bounding volume tests become more expensive and the storage requirement for the bounding volume increases. Typically, bounding volumes are computed in preprocessing step and, as necessary, transformed with the bounded objects at runtime to match the objects' movements.

Collision systems work almost exclusively with convex objects because they have certain properties that make them highly suitable for intersection testing. The only disadvantage of the linear programming approach is the limitation of use with convex objects only. This is due to the fact that convex objects help the algorithms to converge quickly and to report collision if it exists. This little obstacle is easily overpassed by today's techniques of convex decomposition[22].

The multiple purpose of a collision detection algorithm is an important issue. This is widely seen in the linear programming approach shown in section 4.1. This algorithm is used as a collision detection procedure and as a penetration depth vector calculator. This vector is important in predicting collision especially for motion planning problems. Some algorithms used for collision detection are based on the assistance of the hardware especially on the graphics processing units. They were not discussed in this paper because they are only used in a very restriced domain. Finally, even that a huge number of collision detection procedures exist nowadays, this area of research is still wide open for new innovations.

References

[1] O. O'Reilly, P. Papadopoulos, G. Lo, P. Varadi. "Models of Vehicular Collision: Development and Simulation with Emphasis on Safety". Technical Report UCB-ITS-PRR-98-10, University of California, Berkly, 1998.

[2] H. Cakmak, U. Kuhnapfel. "Karlushe Endoscopic Surgery Trainer for Minimally Invasive Surgery in Gynaecology". Proceedings of 13^th International Congree on Computer Assisted Radiology and Surgery, France, June 1999.

[3] S. Cotin, H. Delingette, N. Ayache. "Real-time Elastic Deformations of Soft Tissues for Surgery Simulation". IEEE Transactions on Visualization and Computer Graphics, volume 5 (1), 1998.

[4] C. Fares, Y. Hamam, M. Couprie, R. El-Abyad. "Virtual Arthroscopic Surgery Trainer: A Virtual Reality Based Training System for Arthroscopic Surgery". Proceedings of BioMedSim03, University of Balamnd, Lebanon, May 2003.

[5] B. Honzik, Y. Hamam. "Obstacle Avoidance for Non-Point Mobile Robots". Proceedings of 3^rd IMACS Symposium on Mathematical Modeling, pages 887–890, 2000.

[6] S. Oh, H. Kim, K. Wohn. "Collision Handling for Interactive Garment Simulation". Proceedings of VSMM, 2002.

[7] I. Rudomin, J. Castillo. "Realtime Clothing: Geometry and Physics". WSCG 2002 Posters, Czech Republic, pages 45–48, 2002.

[8] C. Koh, Z. Huang. "A Simple Physics Model to Animate Human Hair Modeled in 2D Strips in Real Time". Proceedings of Computer Animation and Simulation, 2001.

[9] K. Chung, W. Wang. "Discrete Moving Frames for Sweep Surface Modeling". Proceedings of Pacific Graphics, Hsinchu, Taiwan, 1996.

[10] J. Klosowski, M. Held, J. Mitchell, H. Sowizral, K. Zikan. "Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs". IEEE Transactions on Visualization and Computer Graphics, volume 4 (1), pages 21–36, 1998.

[11] D. Eberly. "Dynamic Collision Detection using Oriented Bounding Boxes". Technical Report, Magic Software, 2002.

[12] D. Schmalstieg, R. Tobler. "Real-Time Bounding Box Area Computation". Technical Report TR-186-2-99-05, Vienna University, 1999.

[13] F. Ganovelli, J. Dingliana, O. Sullivan. "BucketTree: Improving Detection Between Deformable Objects". Proceedings of Spring Conference in Computer Graphics, Bratislava, 2000.

[14] M. Lin. "Efficient Collision Detection for Animation and Robotics". PhD Thesis, University of California, Berkeley, USA, 1993.

[15] B. Mirtich. "V-Clip: Fast and Robust Polyhedral Collision Detection". ACM Transactions on Graphics, volume 17 (3), pages 177–208, 1998.

[16] J. Cohen, M. Lin, D. Monacha, M. Ponamgi. "I-collide: an Interactive and Exact Collision Detection System for Large-Scale Environments". Proceedings of ACM interactive 3D graphics inproceedings, pages 189–196, 1995.

[17] G. Bergen. "A Fast and Robust GJK Implementation for Collision Detection of Convex Objects". Journal of Graphics Tools, volume 4 (2), 1999.

[18] H. Sammet, R. Webber. "Hierarchical Data Structures and Algorithms for Computer Graphics". IEEE Computer Graphics and Applications, volume 4 (3), pages 46–68, 1998.

[19] I. Palmer, R. Grimsdale. "Collision Detection for Animation using Sphere Trees". Computer Graphics Forum, volume 14 (2), 1995.

[20] J. Youn, K. Whon. "Realtime Collision Detection for Virtual Reality Applications". Proceedings of IEEE Virtual Reality Annual International Symposium, pages 18–22, 1993.

[21] C. Fares, Y. Hamam."Collision Detection Between Virtual Objects: Application in Minimally Invasive Surgery". Proceedings of ESMc03, University of Naples II, Naples, Italy, October 2003.

[22] C. Fares, Y. Hamam. "Collision Detection Between Virtual Objects Using Optimization Techniques", In John Cagnol and Jean-Paul Zolesio, editors, Information processing: Recent Mathematical Advances in Optimization and Control. Presses de l'Ecole des Mines de Paris, 2005.

[23] E. Larsen, E. Gottschalk, M. Lin, D. Monacha. "Fast Proximity Queries with Swept Sphere Volumes". Technical Report, Dept. of Computer Science, University of North Carolina, 1999.

[24] C. Fares, Y. Hamam. “Collision Detection for Virtual Reality Using Ellipsoidal Fitting”. Proceedings of BioMedSim05, Sweeden, 2005.

[25] E. Rimon, S. Boyd. “Obstacle Collision Detection Using Best Ellipsoid Fit”. Journal of Intelligent and Robotic Systems, volume 18, pages 105–126, 1997.

[26] L. Pronzato. “Acceleration of D-Optimum Design Algorithms by Removing Non-Optimal Support Points”. Technical Report I3S/RR-2002-05-FR, Laboratoire I3S Informatiques Signaux et Systèmes de Sophia Antipolis, Mars 2002.

[27] S. Krishnan, A. Patteka, M. Lin, D. Monacha. “Spherical Shell: A Higher Order Bounding Volume for Fast Proximity Queries”. Proceedings of $3^r d$ International Workshop on Algorithmic Foundations of Robotics, pages 177–190, 1998.

[28] M. Held. “ERIT: A Collection of Efficient and Reliable Intersection Tests”. Journal of Graphical Tools, volume 2 (4), pages 25–44, 1997.

[29] D. Eberly. “Intersection of a Sphere and a Cone”. Technical Report, Magic Software, March 2002.

[30] D. Eberly. “Intersection of Cylinders”. Technical Report, Magic Software, November 2000.

[31] L. Guibas, A. Nguyen, L. Zhang. “Zonotopes as Bounding Volumes”. Proceedings of $14^t h$ Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, USA, January 2003.

[32] C. Fares, Y. Hamam. “A Hybrid Algorithm for Collision Detection”. Proceedings of the EuroSim04, ESIEE Paris, France, September 2004.

[33] F. Meyer. “Digital Euclidian Skeleton”. Visual Communication and Image Processing, volume 1360, pages 251–262, 1990.

[34] C. Fares, Y. Hamam.“Proximity Queries Computation Using Optimisation”. Proceedings of ECCO XVII, AUB, Lebanon, June 2004.

About the Author

- Charbel Fares is a Ph.D. student at Laboratoire Algorithmique et Architecture des Systmes Informatiques (LA^2SI), of the Ecole Supérieure d’Ingénieurs en Electronique et Electrotechnique (ESIEE), France. His contact email is c.fares@esiee.fr.

- Yskandar Hamam is a professor at LA^2SI of ESIEE and a member of Laboratoire d’Instrumentation et de Relations Individu Systme (LIRIS), CNRS FRE 2508, of the University of Versailles Saint-Quentin-En-Yvelines (UVSQ), France. His contact email is y.hamam@esiee.fr.